Deep Reinforcement Learning for Simulated Autonomous Driving

Adithya Ganesh, Joe Charalel, Matthew Das Sarma, Nancy Xu

CS 229 Final Project

Abstract

This research applies deep Q-learning to train an agent that can autonomously drive in the Open Racing Car Simulator (TORCS) [1, 2]. An initial proof-of-concept algorithm was implemented in Flappy Bird. We compare the performance of classical reinforcement learning models, as well as deep Q-learning with fully-connected and recurrent neural networks. Using the TensorFlow and Keras software frameworks, we trained fully connected deep neural networks that are able to autonomously drive across a diverse range of complex track geometries.

Background and Motivation

The Open Racing Car Simulator (TORCS) is a modern simulation platform used for research in machine learning and autonomous driving. Training an autonomous driving system in simulation offers a number of advantages, including cost-effective data acquisition otherwise requiring substantial amounts of labor for driving and labeling. Furthermore, a simulation is an effective way to identify and test failure cases (e.g. collision events) of safety-critical control systems, without having to sacrifice physical hardware.

Introduced in 2015, Deep Q-learning has been been shown to successfully learn policies on high-dimensional sensory input, such as in the domain of classic Atari 2600 games by Google's DeepMind [3]. Notable approaches to training autonomous driving agents in TORCS include deep supervised learning with convolutional neural networks (DeepDriving, [4]) and deep reinforcement learning with CNNs (DeepMind, [5]). This research investigates the performance of other agents for TORCS that apply different neural network architectures and optimization algorithms.

Deep Q-Learning Objective Function

Consider a deterministic policy $\mu_\theta : \mathcal{S} \to \mathcal{A}$ parametrized by $\theta \in \mathbb{R}^n$. Let r_t^γ denote the total discounted reward from time-step t onwards:

$$r_t^\gamma = \sum_{k=t}^{\infty} \gamma^{k-t} r(s_k, a_k),$$

where $0 < \gamma < 1$. We can define a performance objective $J(\mu_\theta)$ as the total discounted reward for a given policy, while the Q-function calculates the total discounted reward given a starting state-action pair:

$$J(\mu) = \mathbb{E}[r_1^\gamma | \mu]; \quad Q^\mu(s,a) = \mathbb{E}[r_1^\gamma | S_1 = s, A_1 = a; \mu].$$

If ρ^μ is the discounted state distribution, the deterministic policy gradient theorem [6] states that:

$$\begin{aligned} \nabla_\theta J(\mu_\theta) &= \int_{\mathcal{S}} \rho^\mu(s) \nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)} \mathrm{d}s \\ &= \mathbb{E}_{s \sim \rho^\mu}[\nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)}]. \end{aligned} \tag{1}$$

This formulation allows us to obtain the optimal policy applying a gradient ascent update rule on the objective $J(\mu_\theta)$.

Methods

As a proof of concept, initial experiments applied classical Q-learning to a discretization of Flappy Bird. We characterized the Flappy Bird state by the vertical and horizontal separation of the bird and the next obstacle and the vertical velocity of the bird. We compress this state into $S \subset \mathbb{Z}^3$:

$$S = \{(\Delta y, \Delta x, v_y) \mid |\Delta y| \le 50,\ 0 \le \Delta x \le 40,\ |v_y| \le 9\}$$

Our action is a binary decision whether to jump or not. We reward staying alive each frame with $R = 1$ and penalize crashing with a reward $R = -10000$.

In our continuous state-action space deep Q-learning for TORCS, each state observation captures the velocity, angle relative to the track, distance to the edge of the track, and RPM of wheels of the car in double precision. Our action is a triple specifying the steering angle, acceleration, and braking in double precision. We reward longitudinal velocity, while we penalize transverse velocity and divergence from the median of the track:

$$R = \sum_t |V_{x,t} \cos\theta_t| - |V_{x,t} \sin\theta_t| - |V_{x,t}||x_t - \mu_t|$$

Figure 1: TORCS simulation environment

Training Models

To diminish the effect of highly correlated training data, we train the neural networks with a large replay buffer D that accumulates 100,000 samples. During training, we update our parameters using samples of experience $(s, a, r, s') \sim U(D)$, drawn uniformly at random.

$$L = \mathbb{E}_{(s,a,r,s') \sim U(D)} \left[\left(r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right)^2 \right]$$

Using nonlinear functions to approximate the Q-function often yields instability or divergence [7]. Here we exploit *experience replay* [3] to increase efficiency and mitigate instability by smoothing over changes in the data distribution. When training RNN models we sample blocks of training data from the same episode with length equal to the buffer network depth. To encourage sensible exploration, we use an Ornstein-Uhlenbeck process [5, 8] to simulate Brownian-motion about the car with respect to its momentum. This policy avoids pathologies of other exploration policies in autonomous driving that frequently cause the car to brake and lose momentum.

Neural Network Architectures

Architecture A: Fully-Connected Networks. In our actor-critic approach to learning, the critic model estimates Q-function values while the actor selects actions based on those estimates. The final layer of the actor network determines acceleration, steering, and brake with fully connected sigmoid, tanh, and sigmoid activated neurons, which bound the respective actions within their domains. In the feedforward network, both the actor and the critic have two fully connected hidden layers with relu and linear activations, respectively. In the critic model, the actions are not made visible until the second hidden layer.

Architecture B: Recurrent LSTM Networks. A powerful variation on feedforward neural networks is the recurrent neural network (RNN). In this architecture, connections between units form a directed cycle. This allows the network to model dynamic temporal and spatial dependencies from time series driving data. Long short term memory (LSTM) networks are a robust variant of RNNs that capture long term dependencies between states. We have implemented an LSTM network using TensorFlow and Keras, and are currently evaluating the driving performance of different architectures. The LSTM-cell can be mathematically formulated as follows:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$C_t = f_t * C_{t-1} + i_t * \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_0)$$

$$h_t = o_t * \tanh(C_t)$$

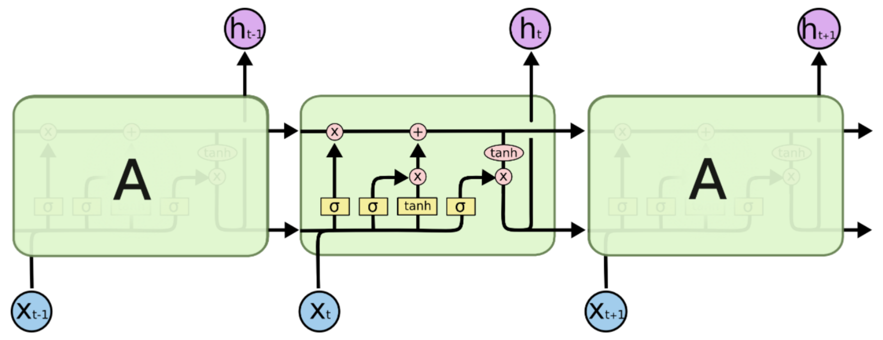


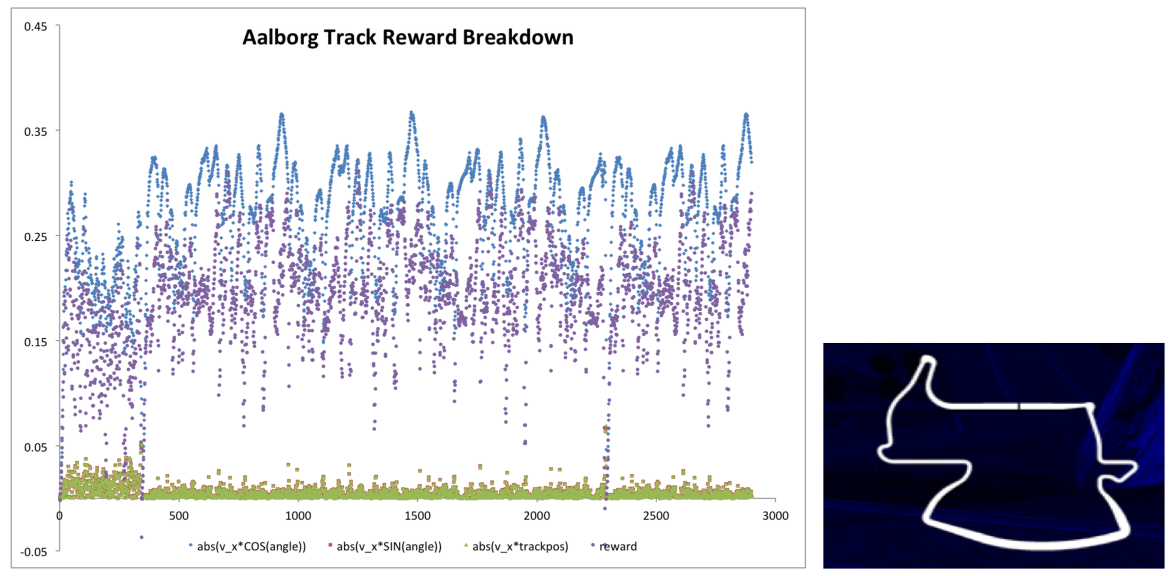
Figure 2: Recurrent Long Short Term Memory (LSTM) Network (image due to Chris Olah)

Results

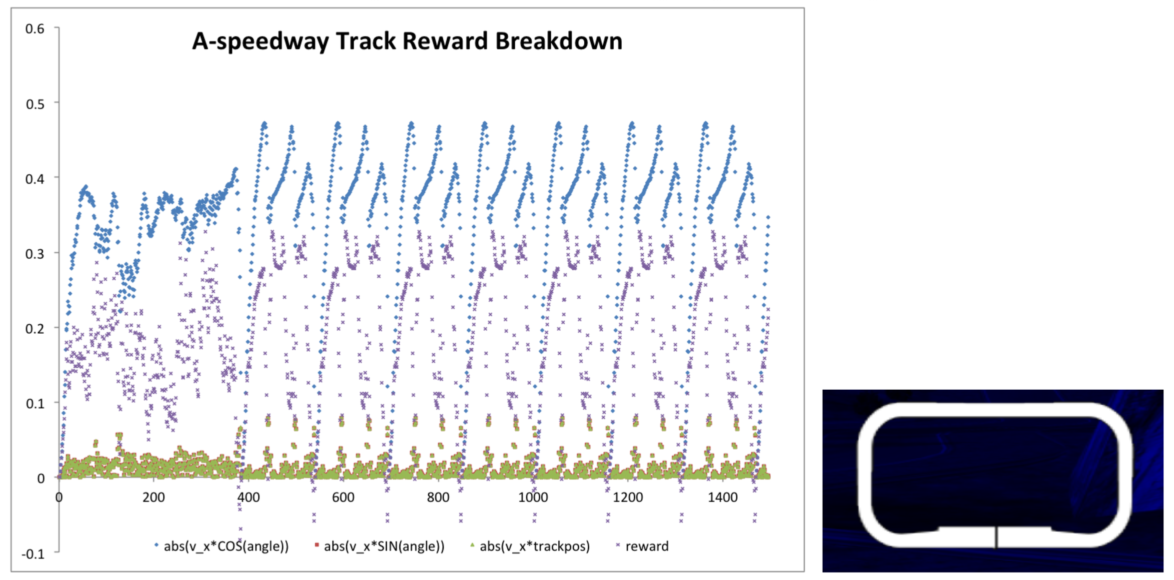
Using the TensorFlow and Keras software frameworks, we trained three-layer fully connected networks with 300 and 600 respective units in each hidden layer. A replay buffer size of 100000 state-action pairs was chosen, along with a discount factor of $\gamma = 0.99$. The Adam optimization algorithm [9] was applied, with a batch size of 32 and learning rates of 0.0001 and 0.001 for the actor and critic respectively.

With this architecture, the resulting agent was capable of navigating a variety of test tracks of highly variable shape in TORCS. We trained the networks on the challenging Aalborg road track (a. in Figure 3). For validation, we tested the networks on the simpler A-Speedway track (b. in Figure 3), as well as the highly irregular Alpine-1 track (c. in Figure 3).

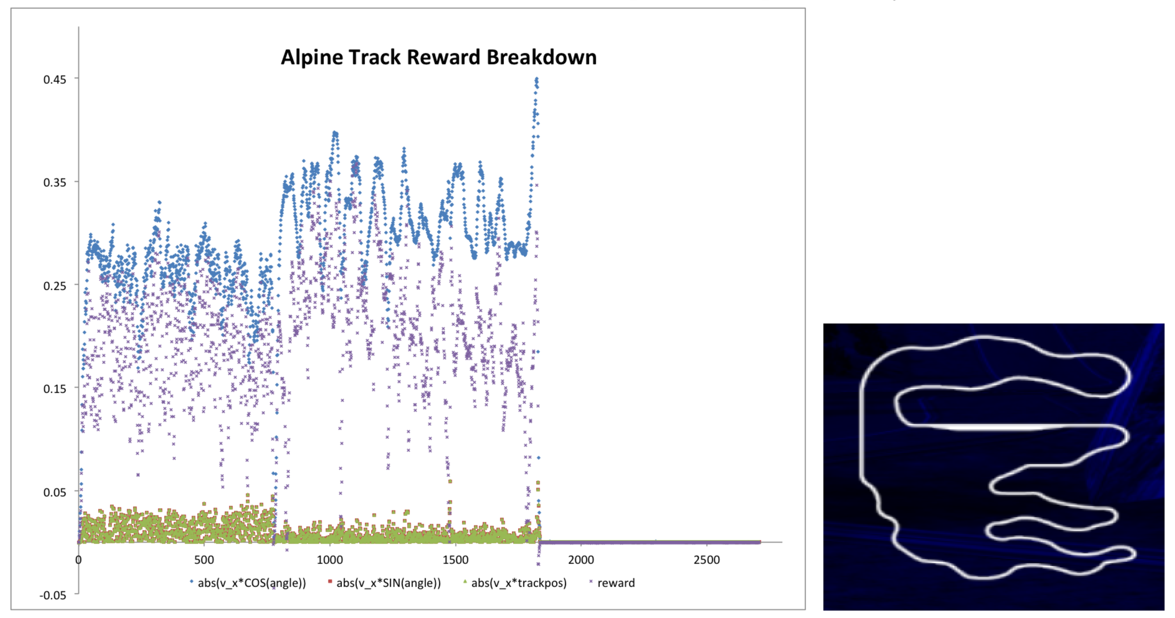
Results: Track Simulation Rewards



a. Training track: Challenging Aalborg track



b. Validation track 1: Simple A-Speedway track



c. Validation track 2: Challenging Alpine-1 track

Figure 3: Rewards obtained during test racing simulations broken down into component terms. The fully connected network architecture was used for these evaluations.

Conclusion and Future Work

We have demonstrated a deep Q-network that can autonomously drive in TORCS, with robustness over diverse environments (including the Aalborg, Alpine-1, and A-Speedway TORCS tracks). Applying reinforcement learning to train an autonomous driving system is an efficient way to simulate model architectures and failure modes without expensive labeling effort and physical hardware.

By modeling long-term state dependencies, we expect LSTM networks to be capable of learning superior policies. We are currently tuning the RNN hyperparameters to achieve optimal convergence. In the future, we would like to test a variety of reward functions, exploration policies, and adaptive gradient descent strategies to optimize the likelihood of near-optimal and rapid convergence. Additionally, we would like to explore the performance of transfer learning to physical hardware and real-world control systems.

Deep Reinforcement Learning for Simulated Autonomous Driving

Adithya Ganesh, Joe Charalel, Matthew Das Sarma, Nancy Xu

CS 229 Final Project

References

[1] Daniele Loiacono, Pier Luca Lanzi, Julian Togelius, Enrique Onieva, David A Pelta, Martin V Butz, Thies D Lonneker, Luigi Cardamone, Diego Perez, Yago Sáez, et al. The 2009 simulated car racing championship. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(2):131–147, 2010.

[2] Daniele Loiacono, Luigi Cardamone, and Pier Luca Lanzi. Simulated car racing championship: Competition software manual. *CoRR*, abs/1304.1672, 2013.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[4] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2722–2730, 2015.

[5] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[6] Guy Lever. Deterministic policy gradient algorithms. 2014.

[7] John N Tsitsiklis and Benjamin Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE transactions on automatic control*, 42(5):674–690, 1997.

[8] Joseph L Doob. The brownian movement and stochastic equations. *Annals of Mathematics*, pages 351–369, 1942.

[9] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[10] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[11] Jens Kober and Jan Peters. Reinforcement learning in robotics: A survey. In *Reinforcement Learning*, pages 579–610. Springer, 2012.

[12] Richard Bellman et al. *Introduction to the mathematical theory of control processes*, volume 2. IMA, 1971.

[13] Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[14] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[15] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[16] Daniel Karavolos. Q-learning with heuristic exploration in simulated car racing. 2013.